系統程式System Programming

蕭銘和
mhhsiao4567@gmail.com

課程目標

- 了解組譯器、編譯器、連結器及作業系統的基本原理，並學習編譯器程式的發展工具，進而能撰寫簡單的編譯器程式。
- 單元主題內容網要
- 1.系統程式簡介　1.何謂系統程式2.電腦系統的演進
- 2.機器語言　1.一般機器結構2.機器語言格式3.電腦動作原理複習
- 3.組譯器　1.符號表資料結構2.字彙分析3.文法分析 4.虛指令處理5.目的碼產生6.one pass組譯器介紹
- 4.巨集　1.簡介2.巨集指令格式及特性3.巨集處理程式介紹
- 5.連結器　1.目的碼簡介2.可重定位連結器
- 6.編譯器　1.編譯器原理、編譯步驟2.資料結構
- 7.作業系統　1.作業系統功能2.程式載入及執行

課程資訊

- 教科書
- 系統程式(Beck：System Software－An Introduction to Systems Programming 3/E)
- (編/譯)者: 王金龍審閱陳健伯·鄭益精·吳坤林· 林婉婷·黃立行·游象勇·姚志岳·陳信宏譯
- 出版年份: 2010
- 發行公司: 培生

- 評量
- 作業(書面+程式)與平時成績(小考+出席) 35%
- 期中考30%
- 期末考35%

系統程式簡介

- 由各種支援電腦運作的程式所組成, 使用者因此可以專注於應用程式。
- 使得使用者開發應用程式（Application）與解決問題，而不需要知道機器的低階運作方式。
- 如 Compiler、 Loader、 Linker、Debugger、 Assembler、Macro Processer、Operating System 等等。

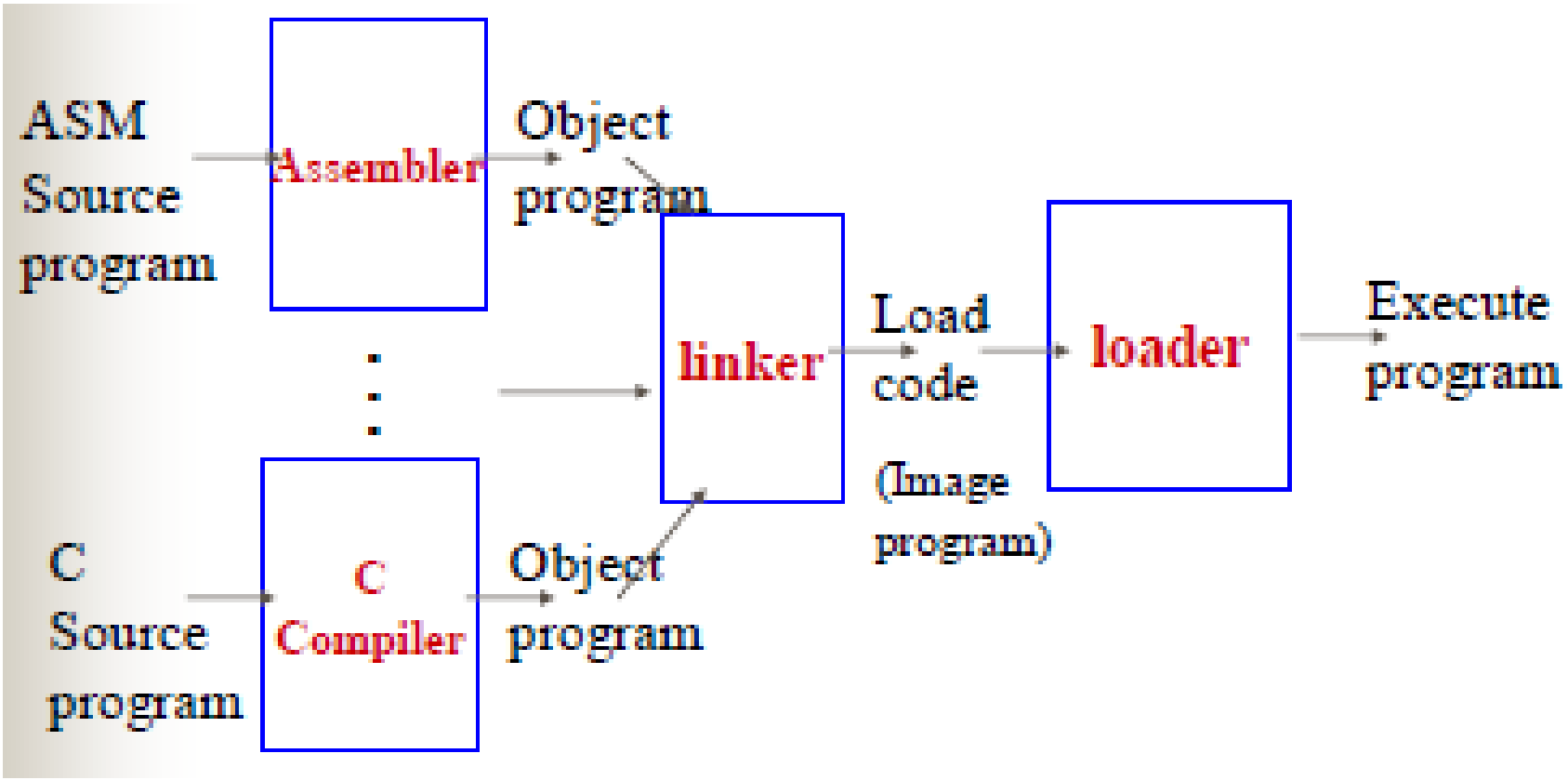
系統程式簡介

- 藉由與電腦的**作業系統**的互動, 操控這些流程
  - UNIX (Linux), DOS: 指令式
  - MacOS, Windows: 視窗式
- 作業系統會幫你處理好機器層面的細節, 你都不必擔心這些問題

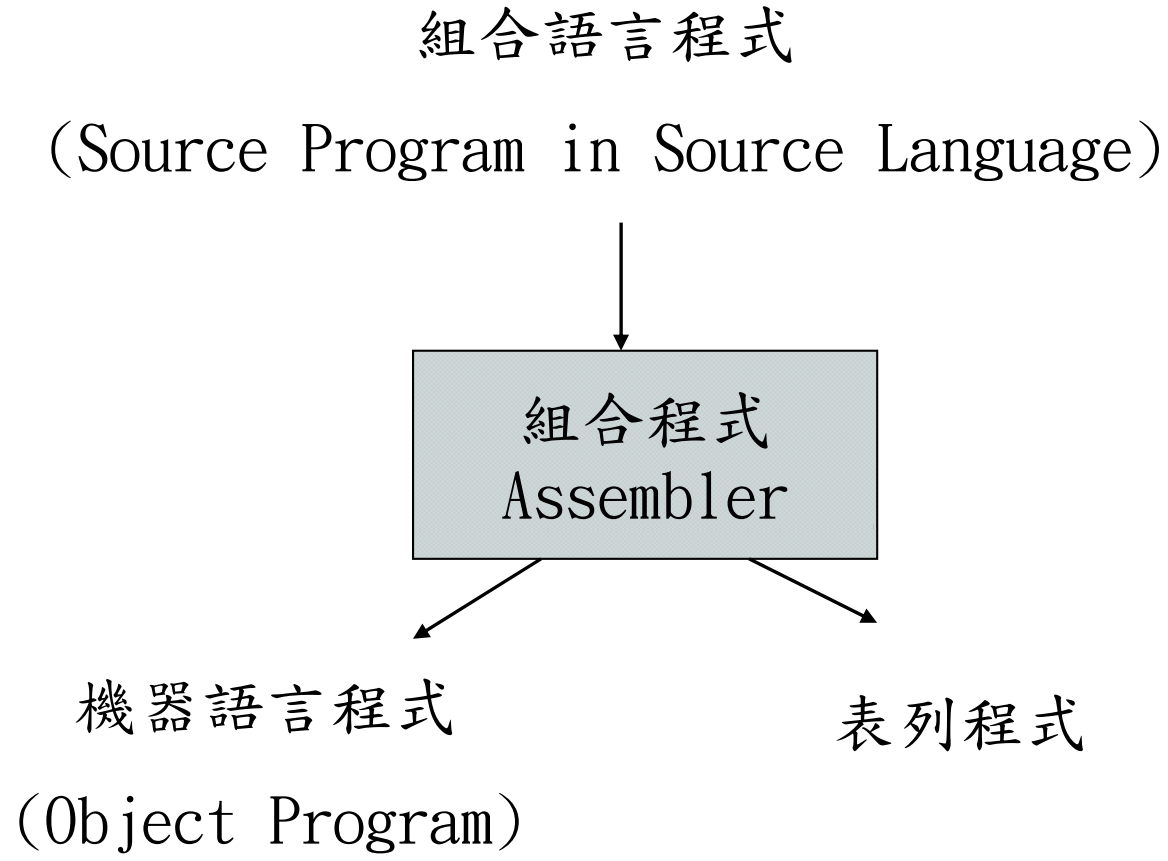
程式編譯過程

1. Preprocess 先處理那些#ifdef #define這些東西並做一些巨集代換
2. Compile 做語意分析，翻譯成組合語言
3. Assemble 翻成機器碼與OS有關的格式，做成relocatable obj檔。
4. Link 找到symbol(函式，變數名)與程式庫(shared obj)中的副程式 ，做成可執行obj檔(executable obj)。

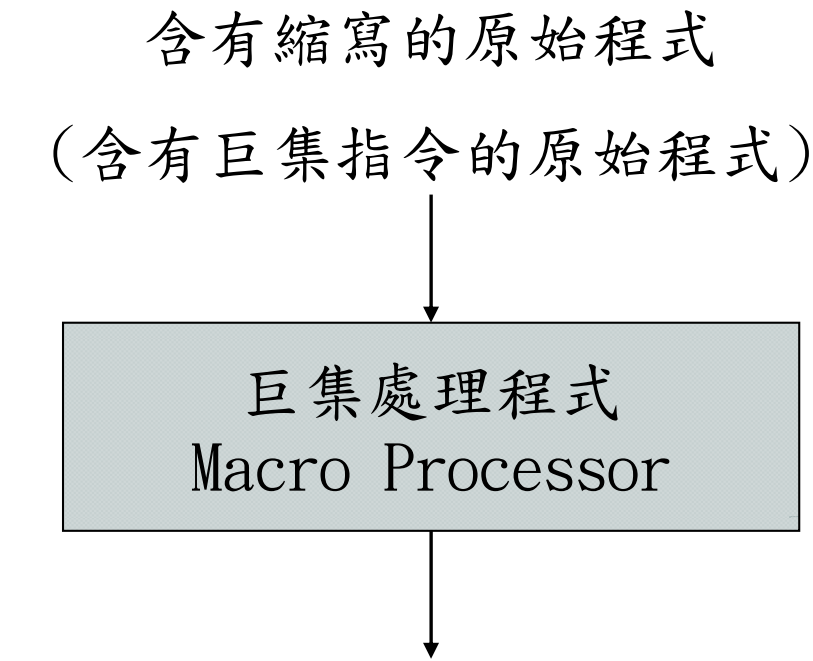
完整程式處理過程


ASM
Source
program
Assembler
Object
program
C
Source
program
C
Compiler
Object
program
linker
Load
code
(Image
program)
loader
Execute
program

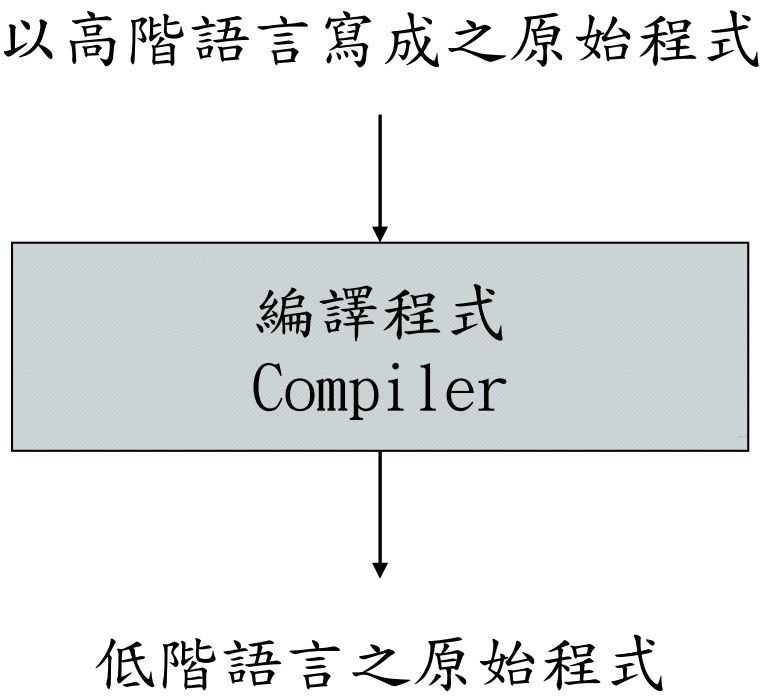
系統程式的形式（1）


組合語言程式
(Source Program in Source Language)
組合程式
Assembler
機器語言程式
(Object Program)
表列程式

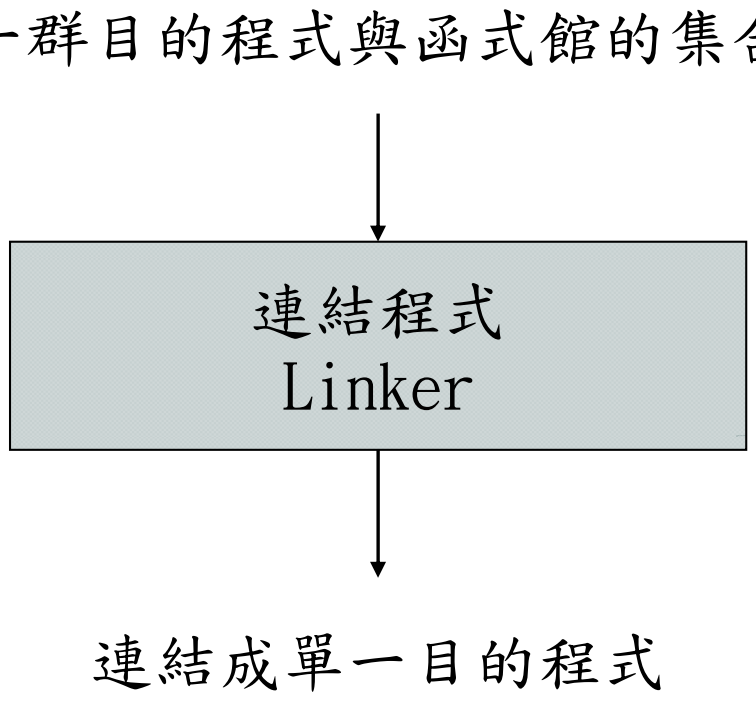
系統程式的形式（2）


含有縮寫的原始程式
（含有巨集指令的原始程式）
巨集處理程式
Macro Processor
不含有縮寫的原始程式
（不含有巨集指令的原始程式）

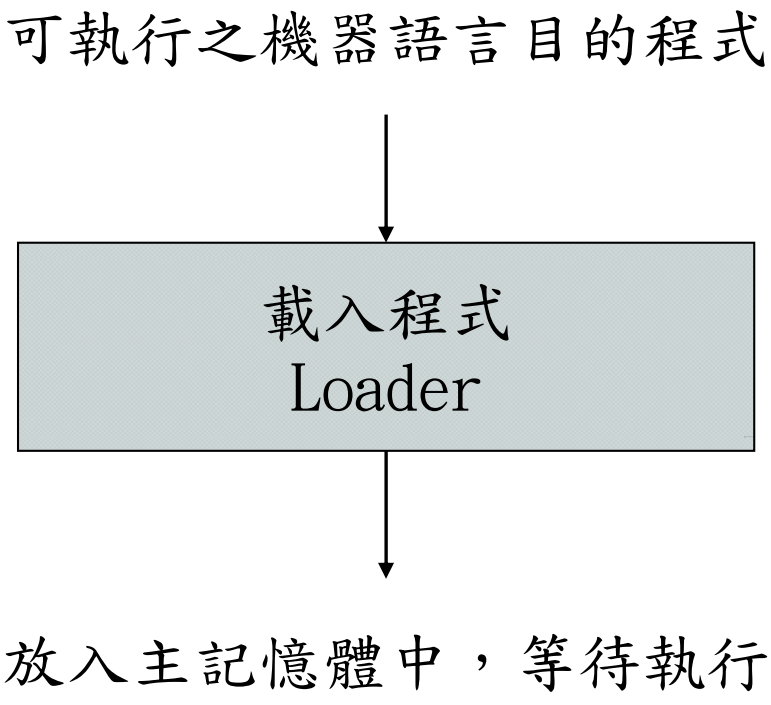
系統程式的形式（3）



系統程式的形式（4）



系統程式的形式（5）



系統程式的形式（6）

- 作業系統：包含了
  - 檔案管理程式
  - 記憶體管理程式
  - 輸出輸入程式
  - 命令解譯程式（Shell）
  - 中斷處理程式
  - ……等等。

系統程式與應用程式

- 應用程式（Application）
  - 是以電腦為工具，用來解決某些問題。
- 系統程式（System Program）
  - 是用來支援使用者對電腦的使用與操作。
  - 與機器的結構有關，但與機器的特性無關。
- 本課程將以 Simplified Instructional Computer（SIC、SIC/XE）系列的電腦作為系統程式的討論平台。

應用程式的範例

- 例如
  - 瀏覽器
  - 排版軟體
  - 多媒體
  - 繪圖軟體
  - ……等等。

程式設計的考量

- 應用程式的設計
  - 考慮與探討應用程式的設計、製作，與維護
- 系統程式的設計
  - 考慮與探討如何設計、製作，與維護以計算機為基礎的資訊處理系統（Computer based Information Processing System）的核心部分。

第1章 背景

內容

- 第1.1 節將說明系統軟體和本書的整體架構。
- 第1.2 節開始探討系統軟體和本書所使用的機器架構之間的關係。
- 第1.3 節將描述具備基本軟體概念的「簡化指令電腦」（Simplified Instructional Computer，SIC）。
- 第1.4 節和第1.5 節展示許多種類的電腦架構範例。

1.1 簡介

- 本節主要是介紹系統軟體（system software）的設計和實作概念。
- 系統軟體是由許多支援電腦運作的程式所組成的，此項軟體可以讓使用者只需關注應用程式或問題的解決方案，而不必知道電腦內部的運作細節。

組合語言

- 組合語言（assembler language）撰寫程式，並且使用巨集指令（macro instructions）來讀取或寫入資料， 或是執行其他的高階功能。
- 然後， 使用具備巨集處理器（macro processor）的組譯器（assembler），將這些程式轉換成機器語言。轉換後的機器碼，可藉由載入器或連結器以置入系統中準備執行，並可以使用除錯器
- 來偵測程式的錯誤。

本書的主題

- 組譯器
- 載入器
- 連結器
- 巨集處理器
- 編譯器
- 作業系統
- 資料庫管理系統
- 文字編輯器以及互動式除錯系統

1.2 系統軟體與機器架構

- 系統軟體和應用軟體最大的不同，就是與機器的相關性
- 系統程式是為了要支援電腦的運作並且供電腦本身的使用，而不是支援特定的應用
- 系統程式通常與其執行機器的架構是有密切的關係: 組譯器, 編譯器,作業系統
- 有許多的系統軟體並不是與其所支援電腦系統的類型有直接的關係。例如，組譯器的一般設計和邏輯

簡化指令電腦(SIC)

- SIC 是一種假想的電腦，其設計包括了大部分真實機器中常見的特性，而避免掉一些不適當的獨特性質。

主要的章節

- 1. 在任何此類軟體的範例中，都能找到的基本功能。
- 2. 與機器架構有緊密關係的特性。
- 3. 在實作這種類型軟體時，其他與機器無關的共通特性。
- 4. 建構特殊軟體的主要設計考量。例如，單階段（single-pass）或是多階段（multi-pass）處理。
- 5. 真實機器上的實作範例，著重不常使用的軟體特性，以及與機器相關的特性。

1.3 簡化指令電腦（SIC）

- 可以代表大多數硬體所具備的特性和概念，而且可以避免真實機器上大部份的特殊性質。
- 二種不同的版本：標準版本和XE 版本

1.3.1 SIC 機器架構

- 記憶體
  - 記憶體是由長度為八個位元的位元組（bytes）所組成；連續三個位元組形成一個字組（word，24 個位元）
  - 所有的位址都是位元組位址（byte address）；字組的位址是以最低位之位元組的位址來表示
  - 電腦記的憶體共有32,768（215）個位元組。

1.3.1 SIC 機器架構

- 暫存器

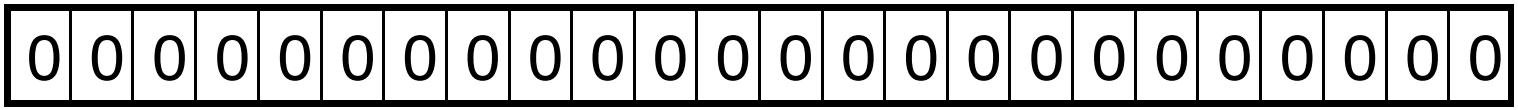
助記符號	編號	特定用途
A	0	累加暫存器（Accumulator）；用在算數運算上
X	1	索引暫存器（Index register）；用在定址上
L	2	連結暫存器（Linkage register）；跳到副程式（Jump to Subroutine, JSUB）指令會將返回的位址儲存在此暫存器中
PC	8	程式計數器（Program counter）；儲存下一個要取出來執行指令的位址
SW	9	狀態字組（Status word）；儲存不同的資訊，包括條件碼（Condition Code, CC）

1.3.1 SIC 機器架構

- 資料格式
  - 整數是以24 位元的二進制值來表示；並以二的補數來表示其負數。
  - 字元是以8 位元的ASCII 碼來表示（參考附錄B）。
  - 在標準版的SIC 中，沒有硬體的浮點運算器。

整數

- 24位元



- 2的補數表示法

Integer		2's Complement
Signed	Unsigned	
5	5	0000 0101
4	4	0000 0100
3	3	0000 0011
2	2	0000 0010
1	1	0000 0001
0	0	0000 0000
-1	255	1111 1111
-2	254	1111 1110
-3	253	1111 1101
-4	252	1111 1100
-5	251	1111 1011

1.3.1 SIC 機器架構

- SIC 指令格式
  - SIC的所有機器碼指令都為底下的24位元格式

- 旗標位元x用來表示索引位址模式

SIC 位址模式

模式	指定	目標位址的算法
直接	x=0	目標位址=位址
索引	x=1	目標位址=位址+(X)

1.3.1 SIC 機器架構

- 指令集
  - 載入和儲存暫存器（LDA、LDX、STA、STX等）
  - 整數算數運算指令（ADD、SUB、MUL、DIV）
  - COMP指令是比較暫存器A和記憶體中字組的值
  - 條件跳躍指令（JLT、JEQ、JGT）
  - JSUB跳到副程式
  - RSUB則是跳到暫存器L中所儲存的位址

1.3.1 SIC 機器架構

- 輸入和輸出
  - 在標準SIC版本中，輸入和輸出的運作是從暫存器A最右邊的八個位元開始
  - 每個裝置都具有一組八位元的裝置碼
  - SIC系統共有三個I/O指令
    - 裝置測試（Test Device, TD）
    - 讀取資料（Read Data, RD）
    - 或寫入資料（Write Data, WD）

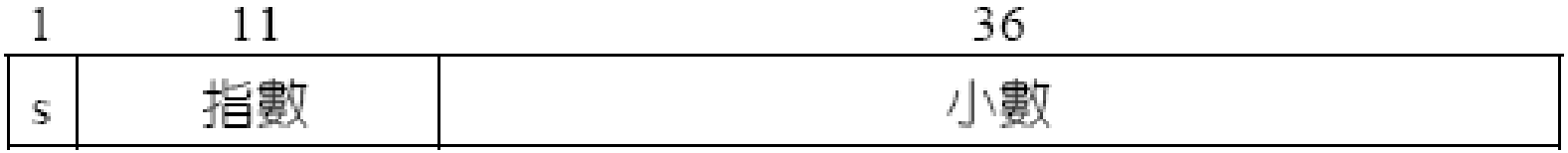
1.3.2 SIC/XE機器架構

- 記憶體
  - 最大可用記憶體為1 M位元組（2^{20}位元組）。
- 額外暫存器

助記符號	編號	特定用途
B	3	基底暫存器；用於定址
S	4	一般工作暫存器 — 沒有特定用途
T	5	一般工作暫存器 — 無特定用途
F	6	浮點累加器（48個位元）

1.3.2 SIC/XE機器架構

- 資料格式



$$f * 2^{(e-1024)}$$

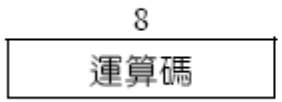
1.3.2 SIC/XE機器架構

- 指令格式：有四種，如下所示
  - 格式一(Operand): 僅有運算子(Operator)，不具運算元
  - 格式二: 運算元為暫存器
  - 格式三: 運算元為相對位址
  - 格式四: 運算元為絕對位址
- 格式如下頁圖所示。

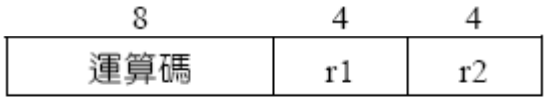
1.3.2 SIC/XE機器架構

- 指令格式

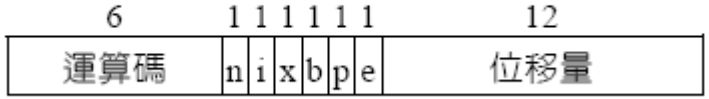
格式 1（一個位元組）：



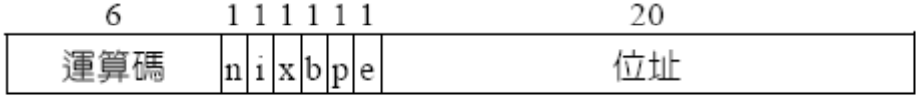
格式 2（兩個位元組）：



格式 3（三個位元組）：



格式 4（四個位元組）：



1.3.2 SIC/XE機器架構

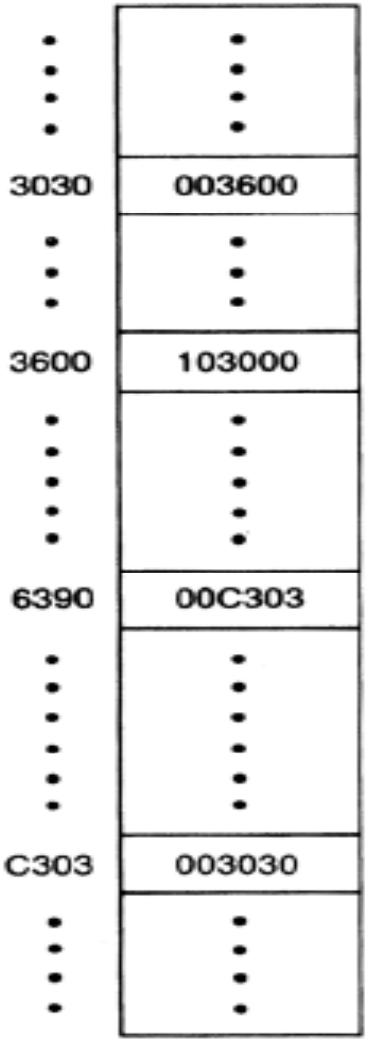
- 定址模式，有兩種：
  - Base Relative
  - Program-Counter (PC) Relative

模式	說明	目的位址計算
相對於基底	b = 1, p = 0	TA = (B) + disp (0 < disp < 4095)
相對於程式計數器	b = 0, p = 1	TA = (PC) + disp (-2048 < disp < 2047)

1.3.2 SIC/XE機器架構

- 其他選項
  - e = 1：Format 4 Instruction
  - e = 0：Format 3 Instruction
  - (b, p) = (0, 0)：Direct Addressing
  - x = 1：Indexed Addressing
  - (i, n) = (1, 0)：Immediate Addressing
  - (i, n) = (0, 1)：Indirect Addressing
  - i = n：Simple Addressing

1.3.2 SIC/XE機器架構



(B) = 006000
(PC) = 003000
(X) = 000090

機器指令

Hex	二進制								目的位址	載入暫存器 A 的值
	op	n	i	x	b	p	e	disp/地址		
032600	000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0	0000 0011 0000	3030	103000
010030	000000	0	1	0	0	0	0	0000 0011 0000	30	000030
003600	000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

1.3.2 SIC/XE機器架構

- 指令集
  - 針對新的暫存器（LDB、STB 等等）進行存取資料，以及執行浮點算數運算（ADDF、SUBF、MULF、DIVF）。
  - 一些暫存器對暫存器的算數運算（ADDR、SUBR、MULR、D1VR）
  - 「監督呼叫指令」（supervisor call, SVC）

1.3.2 SIC/XE機器架構

- 輸出和輸入
  - SIC中的I/O指令，同樣適用於SIC/XE中。
  - 當CPU正在執行其他指令時，還可以用一些I/O通道（I/O channels）繼續執行輸入和輸出的動作。
  - SIO、TIO和HIO指令可以用來啟始、測試和暫停I/O通道的運作。

指令集(Instruction Set) (1)

- 有關算數運算的指令：例如
  - (18)　　ADD　　m
    - A <- (A) + (m..m+2)
  - (58)　　ADDF m
    - F <- (F) + (m..m+5)
  - (90)　　ADDR R1, R2
    - R2 <- (R2) + (R1)

指令集(Instruction Set) (2)

- 有關邏輯運算的指令：例如
  - (40)　　AND　　m
    - A <- (A) & (m..m+2)
  - (44)　　OR　　m
    - A <- (A) | (m..m+2)

指令集(Instruction Set) (3)

- 有關資料傳送的指令：例如
  - (00) LDA m
    - A <- (m..m+2)
  - (68) LDB m
    - B <- (m..m+2)
  - (0C) STA m
    - (m..m+2) <- A
  - (B4) CLEAR R1
    - R1 <- 0

指令集(Instruction Set) (4)

- 有關輸出輸入的指令：例如
  - (E0)　　TD　　m
    - 測試 device
  - (D8)　　RD　　m
    - 讀出 device
  - (DC)　　WD　　m
    - 寫入 device

指令集(Instruction Set) (5)

- 有關跳躍與控制的指令：例如
  - (3C)　　J　　m
  - (30)　　JEQ　　m
  - (48)　　JSUB　　m
    - 跳到副程式
  - (4C)　　RSUB
    - 結束副程式，回到主程式。

指令集(Instruction Set) (6)

- 其他型態的指令：例如
  - (2C) TIX m
    - X <- (X) + 1; (X) : (m..m+2)
  - (B8) TIXR R1
    - X <- (X) + 1; (X) : (R1)
  - (B0) SVC n
    - SVC; (n) = n

SIC 輸出與輸入

- TD 測試未指裝置是否準備好傳送或接收, 結果設定條件碼CC, < 代表已經準備好, = 表示還沒準備好
- 程式必須等待裝置準備好, 才能執行 RD (讀取資料) 或 WD (寫入資料) 指令

SIC 程式設計範例

- 定義資料儲存空間的方法
  - WORD 會保留一個字組的空間 (常數)
  - RESW 會保留一個以上的字組空間供程式使用 (變數)
  - BYTE, RESB 類似, 空間大小僅為一個位元組

1.3.3 SIC程式設計範例

```
        LDA     FIVE        LOAD CONSTANT 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDCH    CHARZ       LOAD CHARACTER 'Z' INTO REGISTER A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
         .
         .
         .
ALPHA   RESW    1           ONE-WORD VARIABLE
FIVE    WORD    5           ONE-WORD CONSTANT
CHARZ   BYTE    C'Z'        ONE-BYTE CONSTANT
C1      RESB    1           ONE-BYTE VARIABLE
```

(a)

```
        LDA     #5          LOAD VALUE 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDA     #90         LOAD ASCII CODE FOR 'Z' INTO REG A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
         .
         .
         .
ALPHA   RESW    1           ONE-WORD VARIABLE
C1      RESB    1           ONE-BYTE VARIABLE
```

(b)

圖 1.2　資料搬移運算範例 (a) SIC (b) SIC/XE

SIC 程式設計範例 – 給值

- 上頁程式 (課本圖1.2a) 相對應的 C 語言

```
int ALPHA = 5;
char C1 = 'Z';
```

1.3.3 SIC程式設計範例

```
        LDS     INCR        LOAD VALUE OF INCR INTO REGISTER S
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADDR    S,A         ADD THE VALUE OF INCR
        SUB     #1          SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADDR    S,A         ADD THE VALUE OF INCR
        SUB     #1          SUBTRACT 1
        STA     DELTA       STORE IN DELTA
        .
        .
        .
.                           ONE WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

(b)

圖 1.3　算數運算範例 (a) SIC (b) SIC/XE

SIC 指令集

- 整數運算 ADD, SUB, MUL, DIV
  - ADD – 將數值加入暫存器 A
  - SUB – 將暫存器 A 減去數值
  - MUL – 將數值乘入暫存器 A
  - DIV – 將暫存器 A 除以數值

1.3.3 SIC程式設計範例

```
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     DELTA       STORE IN DELTA
          .
          .
          .
ONE     WORD    1           ONE-WORD CONSTANT
.                           ONE-WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

(a)

圖 1.3　算數運算範例 (a) SIC (b) SIC/XE

SIC 程式設計範例 – 數值運算

- 上頁程式 (課本圖1.3a) 相對應的 C 語言

```
int         ONE, ALPHA, BETA, GAMMA,
            DELTA, INCR;
BETA = ALPHA + INCR – ONE;
DELTA = GAMMA + INCR – ONE;
```

SIC 程式設計範例 – 迴圈和索引(1.4a)

```
            LDX     ZERO
MOVECH      LDCH    STR1,X
            STCH    STR2,X
            TIX     ELEVEN
            JLT     MOVECH
            …
STR1        BYTE    C’TEST STRING’
STR2        RESB    11
ZERO        WORD    0
ELEVEN      WORD    11
```

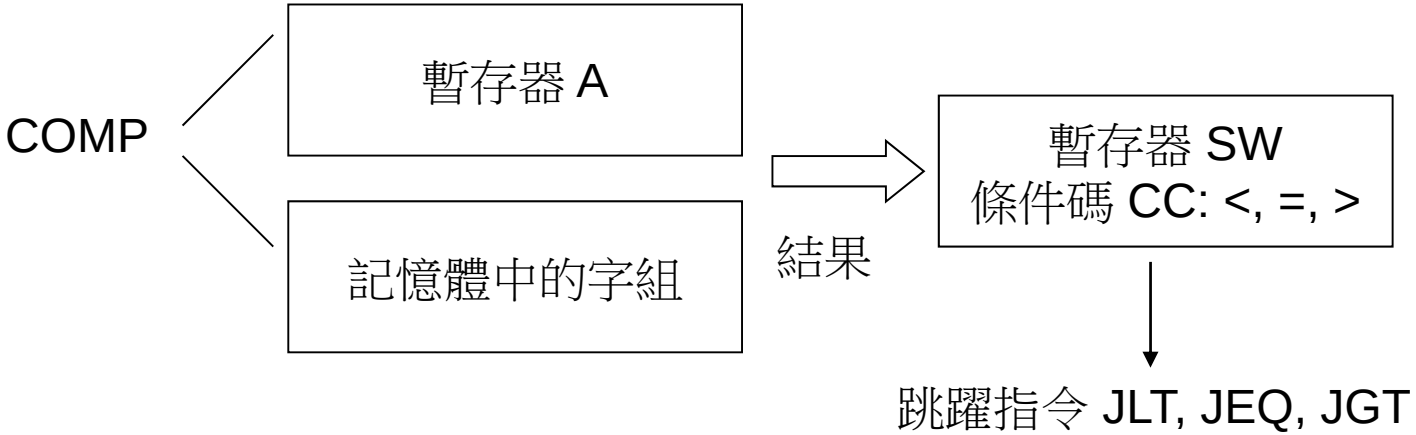
SIC 程式設計範例 – 迴圈和索引

- 上頁程式 (課本圖1.4a) 相對應的 C 語言

```
char *STR1 = “TEST STRING”;
char STR2[11];
strcpy(STR2, STR1);
```

SIC 指令集

- COMP 可比較暫存器 A 的值和記憶體中的字組, 他會設定結果 (<, =, >) 的條件碼 CC
- 條件跳躍指令可以測試CC的設定,根據CC的值來跳躍 (JLT, JEQ, JGT)



SIC 索引

- SIC 指令前可以加索引, 讓跳躍指令使用, 例:

```
        LDA     ALPHA   ;無索引
LABEL1  LDA     ALPHA   ;有索引
```

1.3.3 SIC程式設計範例

```
            LDA     ZERO        INITIALIZE INDEX VALUE TO 0
            STA     INDEX
ADDLP       LDX     INDEX       LOAD INDEX VALUE INTO REGISTER X
            LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
            ADD     BETA,X      ADD WORD FROM BETA
            STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
            LDA     INDEX       ADD 3 TO INDEX VALUE
            ADD     THREE
            STA     INDEX
            COMP    K300        COMPARE NEW INDEX VALUE TO 300
            JLT     ADDLP       LOOP IF INDEX IS LESS THAN 300
            .
            .
            .
INDEX       RESW    1           ONE-WORD VARIABLE FOR INDEX VALUE
.                               ARRAY VARIABLES--100 WORDS EACH
ALPHA       RESW    100
BETA        RESW    100
GAMMA       RESW    100
.                               ONE-WORD CONSTANTS
ZERO        WORD    0
K300        WORD    300
THREE       WORD    3
```

(a)

1.3.3 SIC程式設計範例

```
        LDS     #3          INITIALIZE REGISTER S TO 3
        LDT     #300        INITIALIZE REGISTER T TO 300
        LDX     #0          INITIALIZE INDEX REGISTER TO 0
ADDLP   LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
        ADD     BETA,X      ADD WORD FROM BETA
        STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
        ADDR    S,X         ADD 3 TO INDEX VALUE
        COMPR   X,T         COMPARE NEW INDEX VALUE TO 300
        JLT     ADDLP       LOOP IF INDEX VALUE IS LESS THAN 300
        .
        .
        .
.                           ARRAY VARIABLES--100 WORDS EACH
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
```

(b)

圖 1.5　索引和迴圈運算範例 (a) SIC (b) SIC/XE

1.3.3 SIC程式設計範例

```
INLOOP    TD      INDEV       TEST INPUT DEVICE
          JEQ     INLOOP      LOOP UNTIL DEVICE IS READY
          RD      INDEV       READ ONE BYTE INTO REGISTER A
          STCH    DATA        STORE BYTE THAT WAS READ
          .
          .
          .
OUTLP     TD      OUTDEV      TEST OUTPUT DEVICE
          JEQ     OUTLP       LOOP UNTIL DEVICE IS READY
          LDCH    DATA        LOAD DATA BYTE INTO REGISTER A
          WD      OUTDEV      WRITE ONE BYTE TO OUTPUT DEVICE
          .
          .
          .
INDEV     BYTE    X'F1'       INPUT DEVICE NUMBER
OUTDEV    BYTE    X'05'       OUTPUT DEVICE NUMBER
DATA      RESB    1           ONE-BYTE VARIABLE
```

圖 1.6　SIC 輸入和輸出動作的範例

SIC 指令集

- 連結副程式 JSUB 跳躍至副程式, 將返回位址放到 L; RSUB返回時會跳到暫存器L存放的位址

1.3.3 SIC程式設計範例

```
          JSUB    READ          CALL READ SUBROUTINE
          .
          .
          .
.                               SUBROUTINE TO READ 100-BYTE RECORD
READ      LDX     ZERO          INITIALIZE INDEX REGISTER TO 0
RLOOP     TD      INDEV         TEST INPUT DEVICE
          JEQ     RLOOP         LOOP IF DEVICE IS BUSY
          RD      INDEV         READ ONE BYTE INTO REGISTER A
          STCH    RECORD,X      STORE DATA BYTE INTO RECORD
          TIX     K100          ADD 1 TO INDEX AND COMPARE TO 100
          JLT     RLOOP         LOOP IF INDEX IS LESS THAN 100
          RSUB                  EXIT FROM SUBROUTINE
          .
          .
          .
INDEV     BYTE    X'F1'         INPUT DEVICE NUMBER
RECORD    RESB    100           100-BYTE BUFFER FOR INPUT RECORD
.                               ONE-WORD CONSTANTS
ZERO      WORD    0
K100      WORD    100
```

(a)

1.3.3 SIC程式設計範例

```
          JSUB    READ          CALL READ SUBROUTINE
          .
          .
          .
.                               SUBROUTINE TO READ 100-BYTE RECORD
READ      LDX     #0            INITIALIZE INDEX REGISTER TO 0
          LDT     #100          INITIALIZE REGISTER T TO 100
RLOOP     TD      INDEV         TEST INPUT DEVICE
          JEQ     RLOOP         LOOP IF DEVICE IS BUSY
          RD      INDEV         READ ONE BYTE INTO REGISTER A
          STCH    RECORD,X      STORE DATA BYTE INTO RECORD
          TIXR    T             ADD 1 TO INDEX AND COMPARE TO 100
          JLT     RLOOP         LOOP IF INDEX IS LESS THAN 100
          RSUB                  EXIT FROM SUBROUTINE
          .
          .
          .
INDEV     BYTE    X'F1'         INPUT DEVICE NUMBER
RECORD    RESB    100           100-BYTE BUFFER FOR INPUT RECORD
```

(b)

圖 1.7　副程式呼叫和儲存輸入之運作範例 (a) SIC (b) SIC/XE

1.4 傳統（CISC）機器

- Complex Instruction Set Computers(CISC) 機器
  - 指令集多而暫存器少。（約十幾個）
  - 變動格式的指令。
  - 多種定址形式。
  - 硬體結構較複雜。
- VAX 的架構
- Intel x86 系列的處理器架構

1.4.1 VAX架構

- 記憶體
  - 八位元的位元組所組成
  - 連續的二個位元組形成一個字組（word）
  - 連續四個位元組形成一個「長字組」（longword）
  - 連續八個位元組形成一個「四字組」（quadword）
  - 連續十六個位元組形成一個「八字組」（octaword）
- 所有VAX的程式都可以在2^32位元組的虛擬記憶體空間（virtual address space）中運作

1.4.1 VAX架構

- 暫存器
  - VAX機器有16個通用暫存器（general-purpose registers），編號從R0到R15。
  - 長度都是32個位元。
  - R15為「程式計數器」
  - R14為「堆疊指標」（stack pointer, SP
  - R13為區段指標（frame pointer, FP
  - R12是參數指標（argument pointer, AP）
  - R6到R11:一般用途
  - R0到R5:一般用途/特殊目的

1.4.1 VAX架構

- 資料格式
  - 整數是以位元組、字組、長字組、四字組或八字組
  - 字元是使用8位元的ASCII碼來儲存。
  - 有四種不同的浮點（floating-point）資料格式，其長度範圍從4到16個位元組。
  - 「聚集式十進制」（packed decimal）
  - 佇列（queues）以及可變長度的位元串（variable-length bit strings）

1.4.1 VAX架構

- 指令格式
  - 可變長度的指令格式
  - 運算碼（一個或二個位元組），以及依據指令型態的不同，最多有六個「運算元描述子」（operand specifiers）所組成的。

1.4.1 VAX架構

- 定址模式
  - 暫存器模式
  - 暫存器委託模式
  - 自動增加和自動減少模式
  - 基底相對的定址模式:程式計數器相對模式
  - 所有的定址模式都可以包含一個索引暫存器（index register
  - 指定間接定址模式
  - 立即運算元和許多特定用途的定址模式

1.4.1 VAX架構

- 指令集
  - 指定運算子型態的字首（prefix）。
  - 指定運算元資料型態的字尾（suffix）。
  - 決定運算元數量的修飾子（modifier，某些指令才具備）。

1.4.1 VAX架構

- 輸入和輸出
  - 藉由I/O裝置控制器（device controller）來完成輸入和輸出的動作。
  - 每個控制器擁有一組控制/狀態和資料暫存器，這些暫存器在實際位址空間（physical address space）上都有特定的對應位置。裝置控制器之暫存器所對應的位址空間，稱之為I/O空間（I/O space）

1.4.2 Pentium Pro架構

- 記憶體
  - 在實體層次上，記憶體是由八個位元的位元組所組成的
    - 連續的二個位元組形成一個字組
    - 連續的四個位元組形成雙字組（double-word，也稱為dword）。
  - 程式設計者通常將x86記憶體視為是一些區段（segments）的集合
    - 位址將包含二個部分：區段的編號以及該區段中的位移量（offset）。

1.4.1 VAX架構

- 暫存器
  - 八個通用的暫存器:EAX、EBX、ECX、EDX、ESI、EDI、EBP、以及ESP。(長度都是32位元長)
  - EAX、EBX、ECX、和EDX通常作為資料處理的用途
  - FLAGS是一個長度為32位元的暫存器，其內容可以包含許多不同旗標位元，
  - 區段暫存器（segment registers），用來指定記憶體中的區段。
    - CS存放現正執行之區段碼的位址
    - SS包含了目前堆疊區段的位址
    - DS、ES、FS、和GS指出資料區段的位址。

1.4.1 VAX架構

- 資料格式
  - x86架構提供了整數、浮點數、字元、以及字串的儲存空間。
  - FPU可以處理64位元之具正負符號的整數。
  - x86架構有三種不同的浮點資料格式。
    - 單精度（single-precision）格式的長度是32位元
    - 雙精度（double-precision）格式的長度是64位元
    - 延伸精度（extended-precision）格式的長度是80個位元

1.4.1 VAX架構

- 指令格式
  - x86機器的所有指令是使用相同的基本格式
    - 前置詞（prefix），
    - 運算碼（opcode，一個或二個位元組）
    - 運算元以及所使用的定址模式

1.4.1 VAX架構

- 定址模式
  - 運算元的值可能本身就位於指令中（立即模式），或是儲存在暫存器中（暫存器模式）。
  - TA =（基底暫存器）+（索引暫存器）*（比例因子）+ 位移量

1.4.1 VAX架構

- 指令集
  - 超過四百種不同的機器指令
  - 每一個指令可能會有零個、一個、二個或三個運算元
  - 暫存器對暫存器
  - 暫存器對記憶體的指令
  - 記憶體對記憶體的指令

1.4.1 VAX架構

- 輸入和輸出
  - 藉由I/O埠將一個位元組、字組、或是雙字組，傳送到EAX暫存器中
  - 輸出指令則是從EAX暫存器中，傳輸一個位元組、字組、或雙字組到I/O埠
  - 利用重複前置詞，可以讓這些指令再單一的運算中就可以傳送一個完整的字串。

1.5 RISC機器

- Reduced Instruction Set Computers
- 標準和固定的指令長度
- 單週期（single-cycle）的指令。
- 記憶體的存取通常只能透過載入和儲存的指令
- 所有的指令都是運作於暫存器與暫存器之間
- 通用暫存器的數目都相當的多
- 機器指令的數目、指令格式、以及定址模式卻是相當的少。

1.5.1 UltraSPARC架構

- 記憶體
  - 記憶體是由長度為八位元的位元組所組成的
  - 連續兩個位元組形成「半字組」（halfword）
  - 連續四個位元組形成「字組」
  - 連續八個位元組則形成「雙字組」
  - UltraSPARC的程式可以使用2^{64}位元組的虛擬記憶體

1.5.1 UltraSPARC架構

- 暫存器
  - 非常大量的暫存器檔案（register file
  - 超過100個以上的通用暫存器
  - 一個程序只能夠存取其中的32個暫存器，編號從r0到r31。
    - 前八個暫存器（r0到r7）是全域式（global） 暫存器

1.5.1 UltraSPARC架構

- 資料格式
  - 整數、浮點數以及字元的儲存空間。
  - 整數是以8、16、32或64位元的二進制值來儲存
  - 三種不同的浮點資料格式
    - 單精度（single-precision）格式的長度是32位元
    - 雙精度（double-precision）格式的長度是64個位元
    - 「四精度」（quad-precision）格式中，有63個位元為浮點值，而15個位元為指數值。
  - 字元是儲存在一個位元組中

1.5.1 UltraSPARC架構

- 指令格式
  - 長度都是32個位元
  - 格式1是用於呼叫（call）指令
  - 格式2是使用在分歧（branch）指令
  - 其餘的指令都是使用格式3，包括了暫存器的載入和儲存，以及三個運算元的算數運算。

1.5.1 UltraSPARC架構

- 定址模式

模式	目的位址計算
PC 相關模式	TA =（PC）+ 位移量 {30 位元，有正負號}
具位移量之暫存器間接模式	TA =（暫存器）+ 位移量 {13 位元，有正負號}
暫存器間接索引模式	TA =（暫存器-1）+（暫存器-2）

1.5.1 UltraSPARC架構

- 指令集
  - 少於100 種的機器指令
  - 唯一可以存取記憶體的就是載入和儲存指令
  - 其他所有的指令都是運作於暫存器與暫存器之間。
  - 採用「管線」（pipelined）方式

1.5.1 UltraSPARC架構

- 輸入和輸出
  - 透過記憶體來達成與I/O 裝置的通訊。
  - 某一段範圍的記憶體位址，在邏輯上是被裝置暫存器所佔用。
  - 每個I/O 裝置都有唯一的位址
  - 針對記憶體中的裝置暫存器位址進行載入或儲存時，就可以啟動相對應的裝置

1.5.2 PowerPC 架構

- **記憶體**
  - 八位元的位元組所組成
  - 連續的兩個位元組形成「半字組」
  - 連續四個位元組形成「字組」
  - 連續八個位元組形成「雙字組」
  - 連續十六個位元組形成「四字組」
  - PowerPC的程式可以使用2^{64}位元組的虛擬記憶體

1.5.2 PowerPC 架構

- 暫存器
  - PowerPC架構中有32個通用暫存器，編號從GPR0到GPR31。
  - 長度都是64個位元
  - 浮點運算可以執行於浮點運算單元，此單元包含了32個64位元的浮點暫存器，以及狀態和控制暫存器。
  - 32位元的條件暫存器
  - 「連結暫存器」（Link Register, LR）
  - 「計數暫存器」（Count Register, CR）

1.5.2 PowerPC 架構

- 資料格式
  - 提供了整數、浮點數和字元的儲存空間
  - 整數可以是8、16、32、或64個位元的二進制數值，而且也支援具正負符號以及不具符號的整數；負數是以二的補數方式來表示。
  - 兩種不同的浮點資料格式。
    - 單精度（single-precision）格式的長度是32位元
    - 雙精度（double-precision）格式的長度是64位元
  - 每個字元佔一個位元組，且使用八位元長的ASCII碼表示。

1.5.2 PowerPC 架構

- 指令格式
  - PowerPC架構中有七種基本的指令格式，其中某些格式還有子格式（subforms）
  - 所有格式的長度都是32個位元
  - 指令必須對齊字組邊界的開始位置（亦即，四的倍數的位元組位址）
  - 指令的前六個位元通常都是運算碼；某些指令格式會有額外的延伸運算碼（extended opcode）欄位。

1.5.2 PowerPC 架構

- 定址模式
  - 一個運算元的值可能是位於指令之上（立即模式），或是位在暫存器中（暫存器直接模式）。
  - 唯一可以存取記憶體的是載入和儲存指令，以及分歧指令。

模式	目的位址計算
暫存器間接模式	TA =（暫存器）
具索引的暫存器間接模式	TA =（暫存器-1）+（暫存器-2）
立即索引的暫存器間接模式	TA =（暫存器）+ 位移量 {16 位元，具正負號}

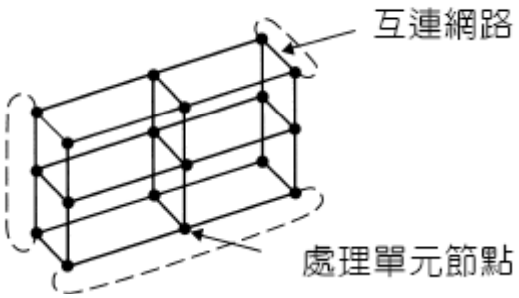
1.5.2 PowerPC 架構

- 指令集
  - 200種機器指令
  - 指令的執行是採用管線（pipeline）的方式
  - 「分歧預測」（branch prediction）的方法

1.5.2 PowerPC 架構

- 輸入和輸出
  - 虛擬位址空間（virtual address space）的區段（segments）會對應到外部位址空間
  - 正常虛擬記憶體的存取方法

1.5.3 Cray T3E架構



整體 T3E 架構

1.5.3 Cray T3E架構

- 記憶體
  - 每個處理單元都有自己的區域記憶體，其容量從64M位元組到2G位元組
  - 系統記憶體（system memory）在實體上是分散於每個PE中的區域記憶體
  - 每一個處理單元的記憶體是由八位元的位元組所組成

1.5.3 Cray T3E架構

- 暫存器
  - Alpha架構包含了32個通用暫存器，編號從R0到R31；R31通常儲存數值0。
  - 每個通用暫存器的長度都是64位元
  - 還有32個浮點暫存器，編號從F0到F31；F31通常儲存數值0。每個浮點暫存器的長度都是64位元。
  - 64位元暫存器，如程式計數器和許多的狀態和控制暫存器。

1.5.3 Cray T3E架構

- 資料格式
  - 整數、浮點數、以及字元的儲存空間。
  - 整數是以長字組或四字組的方式來儲存
  - 而負值是以二的補數來表示

1.5.3 Cray T3E架構

- 指令格式
  - 有五種基本的指令格式，其中有些格式還具有子格式。
  - 所有格式的長度都是32個位元
  - 指令的前六個位元通常是運算碼
  - 某些指令格式還有額外的「功能」欄位（function field）。

1.5.3 Cray T3E架構

- 定址模式

模式	目的位址計算
PC 相對模式	TA =（PC）+ 位移量 {23 位元，具正負號}
具位移量之暫存器間接模式	TA =（暫存器）+ 位移量 {16 位元，具正負號}

1.5.3 Cray T3E架構

- 指令集
  - 130種機器指令
  - 不具備位元組或字組的載入和儲存的指令
  - 記憶體存取的介面並不需要「位移和遮罩」（shift-and-mask）的運作

1.5.3 Cray T3E架構

- 輸入和輸出
  - 透過許多的I/O通道（I/O channels）來執行I/O的動作。
  - 這些I/O通道可以組織成許多種不同的形式，例如整合到銜接「處理節點」（processing nodes）的網路中。
  - 一個系統最多可以將一個I/O通道分配給八個PE使用。
  - dra所有的PE都可以控制和存取所有的通道。

第一章重點

- SIC的架構, 資料格式, 指令集
- SIC/XE的架構, 資料格式, 指令集
- 各類指令的運作方式
- CISC / RISC
- 重點
  - 各式定址模式
  - 寫簡單的SIC程式 (習題為主)
  - 給SIC, SIC/XE程式, 說明其作用
  - CISC / RISC 架構基本知識